Preimage attacks on round-reduced MD5, SHA-1, and SHA-256 using parameterized SAT solver

Oleg Zaikin[1]

[1]ISDCT SB RAS, Lermontov street 134, Irkutsk, 664033, Russia.

Contributing authors: oleg.zaikin@icc.ru;

Abstract

MD5, SHA-1, and SHA-256 are fundamental cryptographic hash functions that produce a hash of fixed size given a message of arbitrary finite size. Their core components are compression functions. The MD5 compression function operates in 4 rounds of 16 steps each, while that of SHA-1 and SHA-256 operate in 80 and 64 rounds, respectively. It is computationally infeasible to invert these compression functions, i.e., to find an input given an output. In 2012, 28-step MD5, 23-round SHA-1, and 16-round SHA-256 compression functions were reduced to SAT and inverted by Conflict-Driven Clause Learning solvers, yet no progress in this area has been made since then. The present paper proposes to construct intermediate inverse problems for any pair of MD5 steps $(i, i+1)$ such that the first problem is very close to inverting i steps, while the last one is almost inverting $i+1$ steps. The same idea works for a pair of sequential rounds in case of SHA-1 and SHA-256. SAT encodings of intermediate problems for MD5, SHA-1, and SHA-256 were constructed, and then a Conflict-Driven Clause Learning solver was parameterized on the simplest of them. The parameterized solver was used to design a parallel Cube-and-Conquer solver that for the first time inverted 29-step MD5, 24-round SHA-1, and 19-round SHA-256 compression functions.

Keywords: Cryptographic hash function, Preimage attack, SAT, CDCL, Algorithm configuration, Cube-and-Conquer

1 Introduction

A cryptographic hash function maps a message of arbitrary finite size to a hash of a fixed size. In the modern digital world, such functions are pervasive for verifying passwords, data integrity, and digital signatures. A secure cryptographic hash function

must be resistant to preimage attacks, i.e., it must be computationally infeasible to invert it by finding a message for a given hash. The present paper is aimed at studying the preimage resistance of the MD5, SHA-1, and SHA-256 cryptographic hash functions. The first two of them are obsolete, while SHA-256 is still secure and widely used.

Given a message, MD5 produces a 128-bit hash by iteratively applying a compression function to 512-bit message blocks. The compression function operates on a 128-bit internal state in 4 rounds of 16 steps each. In each step, the state is modified by mixing with one 32-bit message word. SHA-1 and SHA-256 have similar designs, but differ in the hash length and the way in which compression functions deal with 512-bit message blocks.

Compression functions are the most important components of the considered cryptographic hash functions. Since it is still infeasible to invert MD5, SHA-1, and SHA-256 compression functions, their weakened versions are usually attacked, where some last operations are omitted. In 2007, 26-step MD5 compression function was inverted [1], while for 27- and 28-step versions it was done in 2012 [2]. In 2008 and 2012, 22- and 23-round SHA-1 compression functions were inverted, respectively [2, 3]. In 2012, 16-round SHA-256 compression function was inverted [4]. All these problems were reduced to SAT and solved via SAT solvers based on the Conflict-Driven Clause Learning algorithm (CDCL) [5]. It can be concluded that CDCL solvers are especially efficient in attacking cryptographic hash functions' preimage resistance. However, since 2012, no further progress has been made towards inverting more complex MD5, SHA-1, or SHA-256 compression functions. This paper aims to fill these gaps.

When the number of operations in a cryptographic hash function is reduced, an inverse problem can be further simplified by reducing the number of known hash bits [6, 7]. For example, 29-step MD5 compression function and 64 known output bits instead of 128 can be considered. However, even if this inverse problem is solved, it is not clear if any progress compared to inversion of 28-step MD5 compression function has been made as a result. Indeed, for a certain solver the former problem might be harder than the latter one, while for another solver it might be vice versa. It is also possible to weaken an inverse problem by partially assigning some bits in a message [6, 8]. However, again, it is unclear whether such weakening contributes to progress or not.

This paper proposes a new type of intermediate inverse problems. Consider an arbitrary cryptographic hash function such that its compression function is divided into some basic operations (steps in MD5 or rounds in SHA-1 and SHA-256) and in each operation a k-bit message word m is mixed with an internal state. Note that most existing cryptographic hash functions match this condition. Consider a pair $(i, i+1)$ of operations. The idea is to construct $k-1$ intermediate inverse problems by simplifying operation $i+1$, while the first i operations are left unmodified. In the first problem, in operation $i+1$ the k-bit message word m is replaced by a k-bit word, where $k-1$ bits are constants, while the remaining bit is equal to the corresponding bit in m. In the second problem, $k-2$ bits are constants, while 2 remaining bits are equal to that in m and so on. Finally, in the $(k-1)$th problem, only one bit is constant in the k-bit word. As a result, for a state-of-the-art CDCL solver the first intermediate problem

is slightly harder than inverting i operations, then the hardness gradually increases towards inverting $i+1$ operations.

We construct SAT encodings of 31 intermediate inverse problems between 28- and 29-step MD5 compression functions for one regular hash. Some of them are solved via a state-of-the-art CDCL solver Kissat in a reasonable time. Then we automatically identify Kissat's parameters values that minimize total runtime on several simple intermediate problems, and as a result more intermediate problems are solved by the parameterized solver. Then a parallel Cube-and-Conquer solver based on the same parameterized Kissat inverts 29-step MD5. Kissat is parameterized in the same way on SHA-1 and SHA-256, and as a result the parallel solver inverts 24-round SHA-1, 24.125-round SHA-1, and 19-round SHA-256. Here "24.125" means that the 4th intermediate inverse problem between rounds 24 and 25 of SHA-1 is solved.

In summary, the contributions of the paper are as follows.

- A new type of intermediate inverse problems for a wide class of cryptographic hash functions is proposed.
- An approach for solving an inverse problem via parameterizing a CDCL solver on simple intermediate inverse problems is proposed.
- New SAT encoding of MD5 is proposed.
- An approach for predicting hardness of an inverse problem is proposed.
- For the first time, 29-step MD5 compression function is inverted.
- For the first time, 24-round and 24.125 SHA-1 compression functions are inverted.
- For the first time, 19-round SHA-256 compression function is inverted.

Since MD5 and SHA-1 are obsolete, the results related to them should be regarded as proof of concept of the proposed approach. In contrast, the results for SHA-256 are of practical interest in terms of security.

This paper is an extended version of a paper accepted at the 30th International Conference on Principles and Practice of Constraint Programming (CP 2024) [9]. The main extensions are as follows.

1. The preimage resistance of MD5, SHA-1, and SHA-256 is analyzed, while in [9] only MD5 and SHA-1 were considered.
2. 24.125-round SHA-1 compression function is inverted, while in [9] it was done for at most 24 rounds.
3. Two SAT encodings are built and compared, while in [9] only one SAT encoding was considered.
4. The SAT encoding of intermediate inverse problems from [9] is improved.
5. An algorithm for predicting hardness of inverse problems is additionally proposed.

The paper is structured as follows. Preliminaries on SAT and cryptographic hash functions are given in Section 2. Section 3 proposes intermediate inverse problems. SAT encodings are presented and compared in Section 4. Kissat is applied to intermediate inverse problems in Section 5. Section 6 describes how Kissat is parameterized on intermediate inverse problems. The parameterized versions of Kissat are applied to invert round-reduced MD5, SHA-1, and SHA-256 compression functions in Section 7. Finally, the results and related work are discussed and conclusions are drawn.

2 Preliminaries

This section gives preliminaries on SAT and describes the cryptographic hash functions MD5, SHA-1, and SHA-256.

2.1 Boolean satisfiability problem

Consider a set V of Boolean variables. Using the standard logical operators $\neg, \wedge, \vee$, *propositional Boolean formulae* are defined inductively as follows:

1. $v \in V$ is a propositional Boolean formula;
2. if F is a propositional Boolean formula, then $(\neg F)$ is a propositional Boolean formula;
3. if F and G are propositional Boolean formulae, then $(F \vee G)$ and $(F \wedge G)$ are propositional Boolean formulae.

A propositional Boolean formula is *satisfiable* if there exists a truth assignment to the variables that satisfies it; otherwise it is *unsatisfiable*. The *Boolean satisfiability problem* (*SAT*) is to determine whether a given propositional Boolean formula is satisfiable or not [10]. A *literal* is a Boolean variable or its negation. A *clause* is a disjunction of literals. A propositional Boolean formula is in *Conjunctive Normal Form* (CNF), if it is a conjunction of clauses. SAT is historically the first NP-complete problem [11], but it also has a practical merit since the main complete SAT solving algorithm, *Conflict-Driven Clause Learning* (CDCL) [5], has been successfully applied recently to problems from the following areas: model checking, planning, combinatorics, bioinformatics, and cryptanalysis. *SAT-based cryptanalysis* consists in reducing a cryptanalysis problem to SAT and solving the instance by a SAT solver [12]. In the last two decades, SAT-based cryptanalysis has been successfully applied to stream ciphers, block ciphers, and cryptographic hash functions. In the present paper, SAT-based cryptanalysis is applied to cryptographic hash functions.

2.2 Cryptographic hash function

A *cryptographic hash function* h maps a *message* of arbitrary finite size to a *hash* of finite size [13]. An obligatory property of any cryptographic hash function is that the mapping must be easy to compute, but hard to invert. Consider the following types of resistance.

1. *Collision resistance*: it is infeasible to find any two messages x and x' such that $x \neq x', h(x) = h(x')$.
2. *Preimage resistance*: for any given hash y, it is infeasible to find a message x' such that $h(x') = y$.
3. *Second-preimage resistance*: for any given message x, it is infeasible to find x' such that $x' \neq x, h(x) = h(x')$.

A secure cryptographic hash function must possess all three properties. Usually in practice the collision resistance is the weakest one among them. There are two types of preimage attacks: (i) *practical preimage attack* implies solving an inverse problem,

i.e., finding a preimage (message) for a certain hash; (ii) *theoretical preimage attack* is an algorithm for solving an inverse problem with lower complexity than brute force.

The main component of most cryptographic hash functions is a *compression function* that maps an input of fixed size to an output of shorter fixed size. In order to show that such a cryptographic hash function is not secure, it is sufficient to break its compression function's resistance. In this paper we focus on analyzing the practical preimage resistance of compression functions of the cryptographic hash functions MD5, SHA-1, and SHA-256. In all of them, hash is produced in accordance with the Merkle-Damgård construction [14, 15], i.e., a message is divided into blocks and then a compression function is iteratively applied to the blocks.

2.3 MD5

Message Digest 5 (MD5) is a cryptographic hash function proposed in 1992 [16]. It is a more secure version of MD4 proposed in 1990 [17]. Given a message of arbitrary finite size, *padding* is applied to obtain a message that can be divided into 512-bit blocks. All message blocks are processed by a compression function and finally a 128-bit hash is produced.

Consider the compression function in more detail. Given a 512-bit message block, it produces a 128-bit output. The compression function consists of four rounds of sixteen steps each, and operates by transforming data in four 32-bit registers A, B, C, D. For the first message block, the registers are initialized with constants specified in the standard. Otherwise, the registers are initialized with an output produced at the previous iteration. The message block is divided into sixteen 32-bit words. In each step, three registers are updated by permuting the current values, while one register is updated by mixing one message word, the current values of all four registers, an additive constant, and a result of the previous step. The mixing is partially done by a round-specific function, while additive constants are step-specific. As a result, all sixteen words take part in each round. When all steps are executed, the registers are incremented by the values they had after the initialization, and then a 128-bit output is produced as a concatenation of A, B, C, D.

Consider a pseudocode of an MD5 step in Algorithm 1. Here $i, 1 \leq i \leq 64$ is a step number, $t, 0 \leq t \leq 15$ is a message word index, $\boxplus$ is addition modulo 2^{32}, $\lll$ is the left bit rotation, and K is a step-specific constant.

Algorithm 1 The i-th step of MD5.

Input: current registers' values A, B, C, D; step number i; message word index t; rotate value s; round function Func; constant K.

Output: updated values A, B, C, D.

1: $temp \leftarrow \text{Func}(B, C, D) \boxplus A \boxplus K \boxplus M[t]$
2: $A \leftarrow D$
3: $D \leftarrow C$
4: $C \leftarrow B$
5: $B \leftarrow B + (temp \lll s)$

The round functions are as follows, where all operators are bitwise.

- Round 1, steps 1–16: $F(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$.
- Round 2, steps 17–32: $G(x, y, z) = (x \wedge z) \vee (y \wedge \neg z)$.
- Round 3, steps 33-48: $H(x, y, z) = x \oplus y \oplus z$.
- Round 4, steps 49–64: $I(x, y, z) = y \oplus (x \vee \neg z)$.

Values of t, s, K, and Func for all 64 steps are specified in the standard [16]. Below, values for the first three steps and the last step are given for illustration purposes, while ones for steps 27, 28, and 29 are presented because the corresponding inverse problems will be considered further in the computational experiments.

- Step 1: $0, 7, \texttt{0xd76aa478}, F$.
- Step 2: $1, 12, \texttt{0xe8c7b756}, F$.
- Step 3: $2, 17, \texttt{0x242070db}, F$.
- ...
- Step 27: $3, 14, \texttt{0xf4d50d87}, G$.
- Step 28: $8, 20, \texttt{0x455a14ed}, G$.
- Step 29: $13, 5, \texttt{0xa9e3e905}, G$.
- ...
- Step 64: $9, 21, \texttt{0xeb86d391}, I$.

In 2004, the first MD5 collisions were published [18]. Regardless, it is still preimage resistant and second-preimage resistant in practice. In 2009, the first theoretical preimage attack on MD5 was proposed [19]. Practical preimage attacks on 26-, 27-, and 28-step MD5 compression function are known in the literature [1, 2, 20]. Note that SAT solvers were applied in all these practical preimage attacks. MD5 is deprecated for cryptographic purposes due to existing vulnerabilities.

2.4 SHA-1

Secure Hash Algorithm 1 (SHA-1) was proposed in 1995 as another more secure extension of MD4 [21]. The main differences compared to MD5 are listed below.

1. A 160-bit hash is produced.
2. The compression function operates in 80 rounds and returns a 160-bit output.
3. There are five 32-bit internal registers A, B, C, D, E.
4. New round functions are used.
5. There are four groups of rounds, such that in each group the same round function and round constant are used.
6. Instead of using parts of a message block M directly in rounds, W is used such that $W[t] = M[t]$, $0 \leq t \leq 15$, and $W[t] = (W[t-3] \oplus W[t-8] \oplus W[t-14] \oplus W[t-16]) \lll 1$ if $16 \leq t \leq 79$.

The round functions are listed below. Here all operators are bitwise.

- Group 1, rounds 1–20: $F(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$.
- Group 2, rounds 21–40: $G(x, y, z) = x \oplus y \oplus z$.
- Group 3, rounds 41–60: $H(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$.

- Group 4, rounds 61–80: $I(x, y, z) = x \oplus y \oplus z$.

Consider a pseudocode of an SHA-1 round in Algorithm 2. Recall that a round number i varies from 1 to 80.

Algorithm 2 The i-th round of SHA-1.

Input: current registers' values A, B, C, D, E; round number i; round group number q; round function Func.
Output: updated values A, B, C, D, E.

1: $temp \leftarrow (A \lll 5) \boxplus \text{Func}(B, C, D) \boxplus E \boxplus K[q] \boxplus W[i-1]$
2: $E \leftarrow D$
3: $D \leftarrow C$
4: $C \leftarrow B \lll 30$
5: $B \leftarrow A$
6: $A \leftarrow temp$

A practical collision attack on SHA-1 exists [22], but it still remains preimage resistant and second-preimage resistant. A theoretical preimage attack on 56-round SHA-1 was proposed in [23]. Practical SAT-based preimage attacks on 22- and 23-step SHA-1 compression function are known [2, 3]. Like MD5, SHA-1 is also deprecated for cryptographic purposes.

2.5 SHA-256

Secure Hash Algorithm 2 (SHA-2) was proposed in 2001 as a family of six cryptographic hash functions SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 with hash sizes of 224, 256, 384, or 512 bits [24].

Hereinafter only SHA-256 is analyzed. Its main differences compared to SHA-1 are listed below.

1. A 256-bit hash is produced.
2. The compression function operates in 64 rounds and returns a 256-bit output.
3. There are eight 32-bit internal registers A, B, C, D, E, F, G, H.
4. The same nonlinear functions are applied to update the registers' values each round.
5. W is formed differently.

Consider the following functions, where $\ggg$ denotes the right bit rotation, while all remaining operators are bitwise:

- $\text{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$;
- $\text{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$;
- $\Sigma_0(x) = (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22)$;
- $\Sigma_1(x) = (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25)$;
- $\sigma_0(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \lll 3)$;
- $\sigma_1(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \lll 10)$.

Given a 512-bit message block M divided into 16 32-bit parts, W is formed as follows:

- $W[t] = M[t]$, $0 \leq t \leq 15$;
- $W[t] = \sigma_1(W[t-2]) \boxplus W[t-7] \boxplus \sigma_0(W[t-15]) \boxplus W[t-16]$, $16 \leq t \leq 63$;

Consider a pseudocode of an SHA-256 round in Algorithm 3. Round-specific constants $K[p], 0 \leq p \leq 63$ are specified in the standard [24]. A round number i varies from 1 to 64.

Algorithm 3 The i-th round of SHA-256.

Input: current registers' values A, B, C, D, E, F, G, H; round number i.
Output: updated values A, B, C, D, E, F, G, H.

1: $T1 \leftarrow H \boxplus \Sigma_1(E) \boxplus \mathrm{Ch}(E, F, G) \boxplus K[i-1] \boxplus W[i-1]$
2: $T2 \leftarrow \Sigma_0(A) \boxplus \mathrm{Maj}(A, B, C)$
3: $H \leftarrow G$
4: $G \leftarrow F$
5: $F \leftarrow E$
6: $E \leftarrow D \boxplus T1$
7: $D \leftarrow C$
8: $C \leftarrow B$
9: $B \leftarrow A$
10: $A \leftarrow T1 \boxplus T2$

SHA-256 is still collision resistant, preimage resistant, and second-preimage resistant. A theoretical preimage attack on 52-round SHA-256 compression function is known [25]. In [4], a practical SAT-based preimage attack on 16-step SHA-256 compression function was proposed. Unlike MD5 and SHA-1, SHA-256 is still considered secure. Also, it is widely used in practice, e.g., in Bitcoin mining and to verify authenticity of Debian packages.

3 New type of intermediate inverse problems for cryptographic hash functions

This section first proposes a new type of intermediate inverse problems for cryptographic hash functions from a wide class, and then it describes how such problems can be constructed for MD5, SHA-1, and SHA-256.

3.1 Intermediate inverse problems

Consider an arbitrary cryptographic hash function h with the following features:

1. h has a compression function f;
2. the compression function f is divided into basic operations (e.g., steps in MD5 or rounds in SHA-1 and SHA-256);

3. in each operation, an internal state is mixed with a message word m, where m is either a message part (like in MD5) or a mix of message parts (like in SHA-1 and SHA-256).

Consider a class of cryptographic hash functions based on the Merkle-Damgård construction [14, 15]. This class includes MD4, MD5, the RIPEMD family, SHA-0, SHA-1, and the SHA-2 family. It is clear that all cryptographic hash functions from the class fit the specified requirements. On the other hand, all cryptographic hash functions constructed in accordance with the sponge construction, e.g., SHA-3, do not fit the requirements.

During preliminary experiments, for different triples (a Merkle-Damgård-based cryptographic hash function, a SAT encoding, a CDCL solver) we attempted to invert operation-reduced compression functions, where the last operation was weakened in different ways. It turned out that if the usage of m is omitted in the last operation, then this operation does not really make the inverse problem harder for the CDCL solver. On the other hand, if m is used as usual, yet any other part of the last operation is omitted (including the usage of a nonlinear round function), the inverse problem is almost as hard as the original one. The corresponding details for MD5, SHA-1, SHA-256 will be given in the next subsection.

Based on the above observation, let us propose the following notation. Consider a compression function f that consists of r operations. In each operation, a k-bit message word m is mixed with an internal state. Now consider two reduced versions of f — with the first i and the first $i+1$ operations, where $1 \leq i \leq r-1$. Construct $k-1$ *intermediate operation-reduced compression functions* between them, such that the first i operations are used as usual, yet operation $i+1$ is weakened as follows. In the j-th intermediate function, $1 \leq j \leq k-1$, the word m in operation $i+1$ is replaced by a word m_{weak} such that the rightmost $k-j$ bits in m_{weak} are constants, while the remaining j bits are equal to the leftmost j bits in m. Note that this is not the same as replacing m by m_{weak} in the whole f — if m is used in several operations, then the proposed modification affects only operation $i+1$. In the *j-th intermediate inverse problem* it is needed to find a preimage (input) given an output produced by the j-th intermediate operation-reduced compression function.

The intention behind this approach is that for a state-of-the-art CDCL solver the first intermediate inverse problem will be slightly harder than the inversion of i operations, then the hardness gradually increases towards the inversion of $i+1$ operations.

Note that if for the j-th intermediate inverse problem between operations i and $i+1$ a preimage is found for an output, then with probability $\frac{1}{2^{k-j}}$ the preimage inverts the same output produced by unmodified $i+1$ operations because $k-j$ bits of the corresponding word m can be considered as random bits that can coincide with $k-j$ constant bits with the mentioned probability. This is the first take away of the proposed approach — *if an intermediate inverse problem between operations i and $i+1$ is solved for some output, then with a nonzero probability $i+1$ operations are inverted for the same output.*

3.2 Intermediate inverse problems for MD5, SHA-1, and SHA-256

During preliminary experiments, we tried to weaken step $i+1$ of the MD5 compression function for different values of i (within the second round) by removing $\text{Func}(B, C, D)$ from the addition in the first line of Algorithm 1. However, it did not lead to simpler inverse problems from a state-of-the-art CDCL solver point of view. We also tried to remove other addends in the first line addition, to delete the addend B from the fifth line addition, and to omit rotating in the fifth line. It turned out that the only action that significantly decreases the hardness is removing the addend $M[t]$ in the first line. Moreover, if $M[t]$ is removed, then inverting $i+1$ steps is almost similar to inverting i steps for a CDCL solver.

A pseudocode of an MD5 step weakened according to the proposed idea is presented in Algorithm 4. Here for a k-bit word m the function $\text{Leftmostbits}(m, j)$ returns a k-bit word such that its leftmost j bits are equal to that of m, while the remaining $32-j$ bits are equal to 0. To form 31 intermediate inverse problems with increasing hardness, j should be varied from 1 to 31.

Algorithm 4 The $(i+1)$-th weakened step of MD5.

Input: current registers' values A, B, C, D; step number $(i+1)$; message word index t; rotate amount s; round function Func; constant K; intermediate compression function number j.
Output: updated values A, B, C, D.

1: $weakM \leftarrow \text{Leftmostbits}(M[t], j)$
2: $temp \leftarrow \text{Func}(B, C, D) \boxplus A \boxplus K \boxplus weakM$
3: $A \leftarrow D$
4: $D \leftarrow C$
5: $C \leftarrow B$
6: $B \leftarrow B + (temp \lll s)$

The same pattern was revealed for SHA-1 and SHA-256 — when the usage of W is omitted in round $i+1$, from a CDCL solver point of view an inverse problem becomes almost the same as the inverse of the first i rounds. Again, omitting the usage of a round function (or any other addends) does not really decrease the hardness. Based on these results, intermediate inverse problems for SHA-1 and SHA-256 are formed in the same way as for MD5. Pseudocode of an SHA-1/SHA-256 compression function's round weakened according to the proposed idea is presented in Algorithm 5 and 6.

In the rest of the paper, the j-th intermediate compression function between MD5 steps i and $i+1$, $1 \leq i \leq 63$, is called *(i $\frac{j}{32}$)-step MD5*. Note, that according to this notation $j = 32$ corresponds to $(i+1)$-step MD5. Similarly, *(i $\frac{j}{32}$)-round SHA-1* and *(i $\frac{j}{32}$)-round SHA-256* are denoted.

Algorithm 5 The $(i+1)$-th weakened round of SHA-1.

Input: current registers' values A, B, C, D, E; round number $(i+1)$; round group number q; round function Func; intermediate compression function number j.
Output: updated values A, B, C, D, E.

1: $weakW \leftarrow \text{Leftmostbits}(W[i], j)$
2: $temp \leftarrow (A \lll 5) \boxplus \text{Func}(B, C, D) \boxplus E \boxplus K[q] \boxplus weakW$
3: $E \leftarrow D$
4: $D \leftarrow C$
5: $C \leftarrow B \lll 30$
6: $B \leftarrow A$
7: $A \leftarrow temp$

Algorithm 6 The $(i+1)$-th weakened round of SHA-256.

Input: current registers' values A, B, C, D, E, F, G, H; round number $i+1$; intermediate compression function number j.
Output: updated values A, B, C, D, E, F, G, H.

1: $weakW \leftarrow \text{Leftmostbits}(W[i], j)$
2: $T1 \leftarrow H \boxplus \Sigma_1(E) \boxplus \text{Ch}(E, F, G) \boxplus K[i] \boxplus weakW$
3: $T2 \leftarrow \Sigma_0(A) \boxplus \text{Maj}(A, B, C)$
4: $H \leftarrow G$
5: $G \leftarrow F$
6: $F \leftarrow E$
7: $E \leftarrow D \boxplus T1$
8: $D \leftarrow C$
9: $C \leftarrow B$
10: $B \leftarrow A$
11: $A \leftarrow T1 \boxplus T2$

4 SAT encoding

In this section, two SAT encodings of the considered compression functions are described and compared.

4.1 Description of encodings

Several SAT encodings of the SHA-1 compression function have been proposed so far [2, 3, 6, 26]. However, it seems that at the moment the best one is Vegard Nossum's encoding [6]. Compared to the competitors, it produces CNFs which are easier for CDCL solvers. That is why Nossum's encoding has been used in many studies, e.g., [22, 27]. The paper [22] should be especially highlighted since it presented the first SHA-1 collision.

Recall that in each SHA-1 round a 5-ary addition is performed, see Subsection 2.4. In Nossum's encoding, the column addition algorithm is applied, and each column sum is expressed via a pseudo-Boolean constraint, which is encoded in the clausal form

using the ESPRESSO logic minimizer [28]. Vegard Nossum implemented his encoding in the form of the program SHA1-SAT that is available online[1]. Given a number of rounds and an output, the program produces the corresponding inverse problem for the SHA-1 compression function in the form of a CNF.

We decided to extend SHA1-SAT to support the MD5 compression function. Recall that in each MD5 step a 4-ary addition is performed, so we constructed the corresponding pseudo-Boolean constraint and encoded it in the clausal form using ESPRESSO. As for the round functions (see Subsection 2.3), clausal forms of three of them were taken from the SHA-1 encoding, while the remaining one was constructed manually. The extended version is available online[2].

As for SHA-256, we used the SAT-ENCODING program that is another implementation of Nossum's encoding for MD4, SHA-1, and SHA-256[3]. This program was used in [29] to attack SHA-256.

We also tried the TRANSALG tool [30] to produce SAT encodings. This tool translates algorithms written in a C-like domain-specific language called *TA language* into CNFs. There are three reasons why we chose TRANSALG. First, it is mostly oriented on cryptographic algorithms. Second, it has shown good efficiency when analyzing cryptographic hash functions' resistance [20, 31]. Third, TA programs for MD5, SHA-1, and SHA-256 are already available online[4], so we just took them and made CNFs via TRANSALG version 1.1.6.

4.2 Comparison of SAT encodings

Appendix A contains eight random 256-bit sequences. The first 128/160/256 bits of each sequence were used as a random output for MD5/SHA-1/SHA-256.

Consider the first 21-25 rounds of the SHA-1 compression function and 10 160-bit outputs: 160 zero-bits, 160 one-bits, and eight random outputs described above. For each pair (r, y), where r is a number of rounds and y is a 160-bit output, two CNFs were constructed using TRANSALG and the extended version of SHA1-SAT. Each CNF encodes an inverse problem: given y produced by an r-round SHA-1 compression function, find any its preimage (input). Table 1 shows characteristics of the CNFs for 24 and 25 rounds. It is clear that Nossum's encoding introduces about 2 times fewer variables than that of TRANSALG, yet the number of literals is about 2.5 times higher.

As for MD5, CNFs were constructed for 27-30 steps for 128 zero-bits, 128 one-bits, and eight random 128-bit outputs using TRANSALG and the extended version of SHA1-SAT, see Table 1. Finally, CNFs were constructed for 17-19 rounds of SHA-256 for ten 256-bit outputs using TRANSALG and the extended version of SAT-ENCODING, see Table 1. Similarly to SHA-1, Nossum's encoding provided less variables but more literals.

A state-of-the-art CDCL solver KISSAT version 4.0.1 [32] was run on all constructed CNFs with a time limit of 1 day (24 hours) on a computer equipped with a 16-core CPU AMD Ryzen 3950X and 64 Gb of RAM. This solver was chosen because it has shown excellent results in the last five SAT Competitions. The results are shown as cactus

[1] https://github.com/vegard/sha1-sat
[2] https://github.com/olegzaikin/sha1-sat
[3] https://github.com/saeednj/SAT-encoding
[4] https://gitlab.com/satencodings/satencodings

Table 1: Characteristics of CNFs for round-reduced compression functions.

Compression function	Encoding	Variables	Clauses	Literals
24-round SHA-1	Transalg	8 763	84 795	380 129
	Nossum	4 448	138 764	913 620
25-round SHA-1	Transalg	9 011	87 582	393 724
	Nossum	4 608	144 856	953 513
29-step MD5	Transalg	7 992	57 750	226 908
	Nossum	7 392	95 770	524 232
30-step MD5	Transalg	8 235	59 664	234 470
	Nossum	7 616	99 020	542 144
18-round SHA-256	Transalg	11 834	69 219	240 927
	Nossum	9 070	151 508	899 012
19-round SHA-256	Transalg	12 644	74 662	260 438
	Nossum	9 645	162 574	964 390

plots in Figure 1. For both encodings, all inverse problems were solved in case of 21-22 rounds of SHA-1, 27-28 rounds of MD5, and 17-18 rounds of SHA-256. Additionally, one preimage for 23-round SHA-1 was found when using Nossum's encoding. As a result, this encoding outperformed TRANSALG for all compression functions, yet only in case of SHA-1 the advantage was significant. In the rest of the paper only Nossum's encoding is used in the experiments.

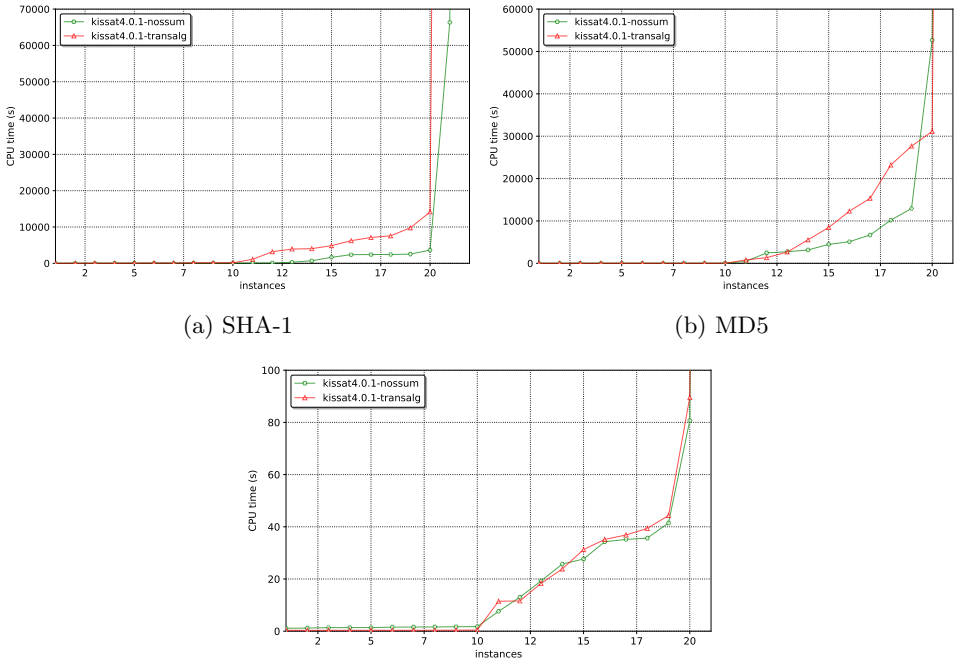
4.3 Encoding intermediate inverse problems

In SHA1-SAT and SAT-ENCODING it should be written explicitly how a new operation is encoded in the clausal form. We modified both tools to maintain intermediate inverse problems. In case of SAT-ENCODING, the extended version is available online as well[5]. Assume that it is needed to encode the j-th intermediate inverse problem between operations i and $i+1$. The first approach is as follows.

1. An additional 32-bit word $mweak$ is introduced in the form of 32 Boolean variables.
2. The equality conditions for the leftmost j bits of $mweak$ and the corresponding j bits of m are added in the form of $j \times 2$ binary clauses[6].
3. The rightmost $32-j$ bits of $mweak$ are assigned to 0 via adding the corresponding unit clauses.
4. 32 Boolean variables of $mweak$ are used instead of m's 32 variables in the clauses that encode the addition in operation $i+1$.

According to the second approach, only $32-j$ new variables are introduced to encode the rightmost $32-j$ bits of the 32-bit $mweak$, while the leftmost j bits of $mweak$ are encoded by the same variables that are used to encode the leftmost j bits of the word m. As a result, no additional binary clauses are added to the CNF since they are not needed anymore. In preliminary experiments the second approach turned

[5] https://github.com/olegzaikin/SAT-encoding
[6] The equality of Boolean variables x and y in the clausal form is $(x \vee \neg y.) \wedge (\neg x \vee y)$



(a) SHA-1

(b) MD5

(c) SHA-256

Fig. 1: KISSAT's runtimes on CNFs that encode inverse problems.

out to be about 10% better runtime-wise, so in the rest of the paper only this approach is used. Note that in [9] only the first approach was implemented.

Characteristics of CNFs for intermediate inverse problems are not presented here since they just slightly differ from the standard ones described in the previous subsection. Namely, there are 32-j more variables and literals in an intermediate CNF.

5 On runtimes of intermediate inverse problems

In the previous section, KISSAT's results on standard inverse problems for the SHA-1, MD5, and SHA-256 compression functions were presented. In this section, we study how KISSAT behaves on intermediate inverse problems produced in accordance with Nossum's encoding. We also show how such runtimes can be used to predict runtimes of hard inverse problems.

5.1 Solving intermediate inverse problems by Kissat

We considered 33 inverse problems for SHA-1: one for 22 rounds, 31 intermediate problems between rounds 22-23, and one for 23 rounds. Similarly to Section 4, we generated 10 instances for each problem — one for 160 one-bits, one for 160 zero-bits, and eight for random outputs, i.e., 330 CNFs in total. Within a time limit of 1 day,

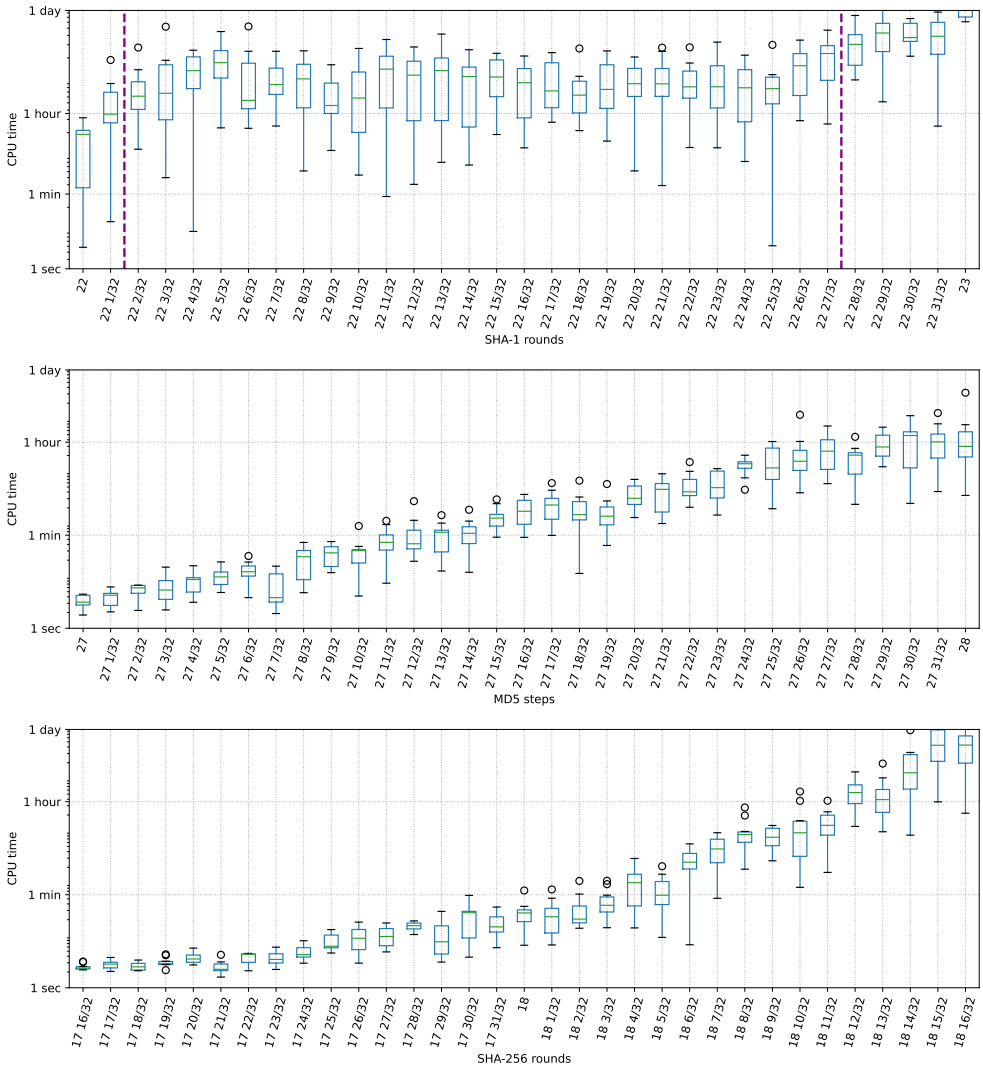


Fig. 2: Boxplots of KISSAT runtimes on intermediate inverse problems.

323 instances were solved on the same 16-core computer as in Section 4. Figure 2 shows the runtimes in the logarithmic scale. The five-number statistics is presented in the form of boxplots: outliers are plotted as small circles, whiskers correspond to the minimum and the maximum (excluding outliers), a median is plotted in green, while a box is plotted from the first quartile to the third quartile. In the boxplots, unsolved instances' values were set to 1 day.

Then 33 inverse problems and the corresponding 330 instances were considered for MD5 and SHA-256 as well — between steps 27-28 and between rounds 17 $\frac{16}{32}$ and 18

$\frac{16}{32}$, respectively. The results are shown in Figure 2. Within the same time limit, all MD5 instances and 325 SHA-256 instances were solved.

According to the results, hardness of the intermediate inverse problems grows almost linearly for MD5 and SHA-256. In case of SHA-1 the pattern is different. The hardness is more or less the same for a series of problems, then a leap is observed and the hardness remains the same for the next series and so on. A leap in hardness means that the median time of any problem from a series is higher than that of the previous series and lower than that of the next series. In the figure, three such series can be seen for SHA-1: between rounds 22 and 22 $\frac{1}{32}$, rounds 22 $\frac{2}{32}$ and 22 $\frac{27}{32}$, and finally rounds 22 $\frac{28}{32}$ and 23. Note that all 7 unsolved instances belong to the last series. In the figure, the described series are divided by dashed vertical lines.

As mentioned in Section 2, it is sufficient to invert any hash to break the preimage resistance of a cryptographic hash function. The same holds for a compression function — it is sufficient to invert any its output. However, in practice this should be a regular output, say all zero-bits or all one-bits, otherwise it may be hard to justify this choice and prove that a random pair of input and output was not picked. In the rest of the paper, 128 one-bits, 160 one-bits, and 256 one-bits are chosen as given outputs for round-reduced MD5, SHA-1, and SHA-256 compression functions, respectively. Further they are called *1-hashes*.

Instances of intermediate inverse problems between rounds 23-24 of SHA-1, steps 28-29 of MD5, and rounds 18-19 of SHA-256 were generated for 1-hashes. Recall that at most 28-step MD5, 23-round SHA-1, and 16-round SHA-256 are inverted in the literature. Note that 17-round SHA-256 can be inverted easily, so it is not considered further in the paper. Four intermediate problems for $j = 16, 19, 20, 22$ were solved by Kissat with a time limit of 1 day in case of SHA-1. Ten MD5 intermediate problems for $j = 1, 2, 3, 4, 5, 7, 8, 10, 12, 14$ were solved. In case of SHA-256, seventeen intermediate inverse problems were solved for $j = 1, \ldots, 16$ and $j = 18$. Therefore (23 $\frac{22}{32}$)-round SHA-1, (28 $\frac{14}{32}$)-step MD5, and (18 $\frac{18}{32}$)-round SHA-256 compression functions are inverted, and this is already a clear progress compared to the literature, but in the next sections we are going further.

The second take away of the approach proposed in Section 3 is as follows: *by solving intermediate inverse problems between operations i and $i + 1$, some progress can be achieved compared to the state of the art, where at most i operations are inverted.*

5.2 Predicting runtimes of hard inverse problems

We decided to apply regression analysis to predict Kissat's runtimes of inverting 1-hash produced by 24-round SHA-1, 29-step MD5, and 19-round SHA-256. For this purpose, intermediate inverse problems between rounds 21-24 of SHA-1, steps 27-29 of MD5, and rounds 18-19 of SHA-256 were considered for 1-hash. Some of the corresponding CNFs and runtimes were taken from the previous subsection, yet the remaining CNFs were generated and Kissat was run on them with a time limit of 1 day. Instances of rounds 17.5-18 of SHA-256 were excluded since they were too simple. In total, runtimes of 97, 65, and 33 instances were analyzed for SHA-1, MD5, and SHA-256, respectively. The runtimes are shown in Figure 3. There is a clear trend that the runtime increases. On the other hand, it does not increase monotonically.

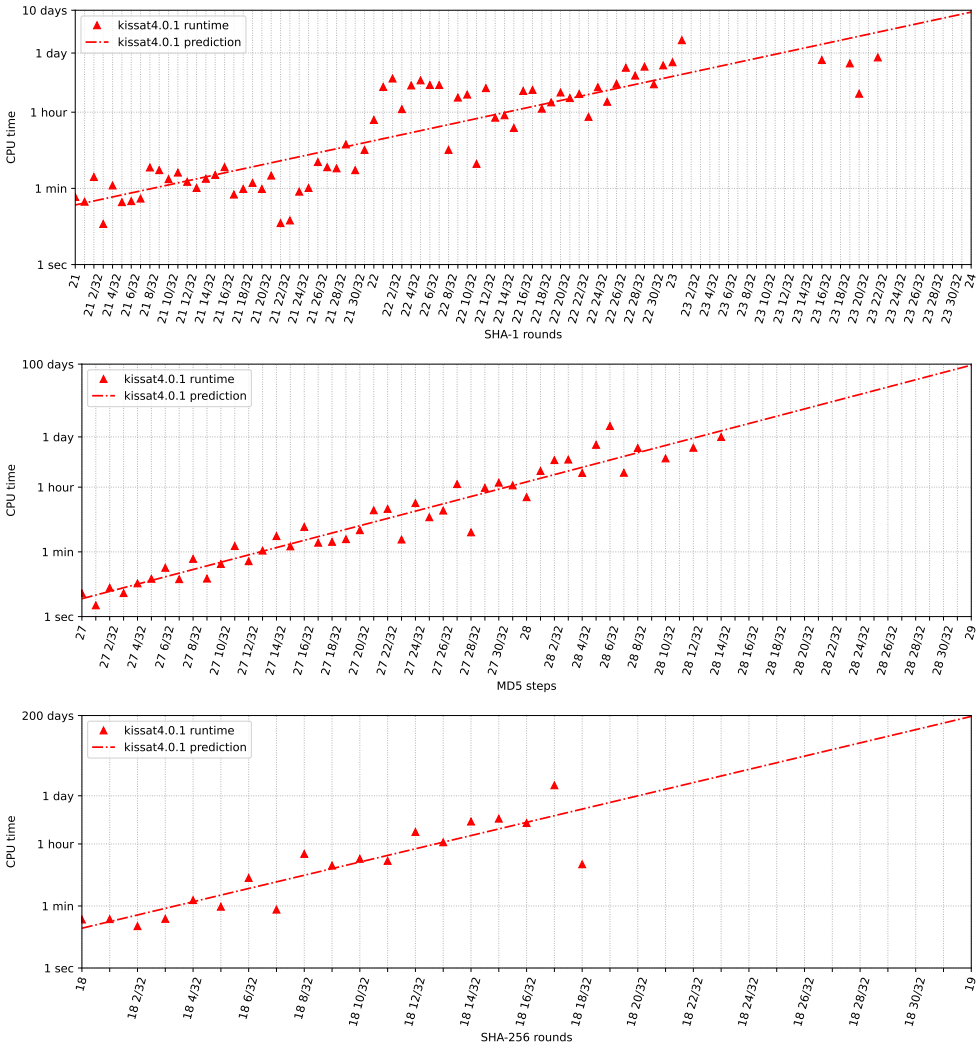


Fig. 3: KISSAT's runtimes on inverse problems for 1-hash and round-reduced MD5, SHA-1, and SHA-256. The y-axis is in the logarithmic scale.

Consider the rightmost runtime in a figure. All runtimes to the left of it are considered as a dataset for regression analysis. For example, in case of SHA-1, the dataset consists of all instances between round 21 and round 23 $\frac{22}{32}$. It is clear that some runtimes are missing. It is well known that missing data negatively impacts regression analysis. Various imputation methods exist which can handle this issue. We applied the following simple method: runtime of the first instance, for which the solver had been interrupted, was assigned to a doubled time limit (i.e., 2 days). It corresponds to PAR-2 (Penalized Average Runtime with a factor of 2 score) that is usually used in SAT Competitions to evaluate solvers' performance.

An exponential relationship between an intermediate inverse problem number x and a runtime y was observed. We decided to calculate the binary logarithm $\log_2 y$ of the runtimes to linearize the relationship and to apply the linear regression to predict runtimes of the unsolved instances. For each compression function, the values a and b in the equation $\log_2 y = a + bx$. were estimated by the ordinary least squares [33]. The corresponding regression lines are shown in Figure 3. As a result, the following runtime predictions were calculated: 9 days to invert 24-round SHA-1; 94 days to invert 29-step MD5; 189 days to invert 19-round SHA-256.

Based on these results, the third take away of the proposed approach can be formulated: *runtime for an unreachable operation can be predicted by extrapolating runtimes of previous intermediate problems.*

6 Parameterizing Kissat by ParamILS on intermediate inverse problems

We put forward a hypothesis that simple intermediate inverse problems can be leveraged to parameterize Kissat to solve hard intermediate problems faster or just solve within a time limit if it is not yet so. On the one hand, parameterizing simplifies problems which are already simple, and it is not guaranteed that it will help solving hard problems. On the other hand, all corresponding CNFs have a very similar structure. We decided to test this hypothesis in practice.

6.1 Approach

Consider an algorithm with parameters, which affect the algorithm's performance. Parameters' values can be integer, real, or categorical. *Algorithm configuration* is the problem of automatically identifying *parameter configuration* (a set of parameters' values) that optimizes the performance of a given algorithm on a set of instances [34]. An algorithm to be parameterized is called *target* algorithm, while an algorithm aimed at solving an algorithm configuration problem is called *algorithm configurator* (or *configurator* for short). A *configuration space* consists of all possible configurations; a configurator begins from a start configuration and searches for better configurations across the configuration space. For each considered configuration, an *objective function* is calculated and the goal is to optimize this function. Usually it is needed to minimize an objective function PARX — the average runtime over all instances whereby unsolved instances count as X times a time limit.

To test the aforementioned hypothesis, the following inputs should be given to a configurator:

1. training set — a set of SAT instances that encode intermediate inverse problems between operations i and $i + 1$;
2. CDCL solver as a target algorithm to be parameterized;
3. solver's configuration space;
4. start configuration;
5. objective function;
6. time limit for the solver;

7. time limit for the configurator.

6.2 Choosing algorithm configurator

The main configurators for CDCL solvers are ParamILS [35], SMAC [36], and GGA [37]. They are based on iterated local search, sequential model-based optimization, and genetic algorithms, respectively. These configurators were used in the Configurable SAT Solver Competition in 2013 and 2014 [38]. During these competitions, each considered parameterized CDCL solver was scored by the best performance it achieved among all configurators.

A recent implementation of GGA is PyDGGA [39] while that for SMAC is SMAC3 [40]. We were choosing between ParamILS and SMAC3 for our experiments because they are free software (unlike proprietary PyDGGA). Finally we chose ParamILS since the current version of SMAC3 does not natively maintain algorithm configuration.

6.3 Configuration space for Kissat

We decided to parameterize Kissat 4.0.1 that was used in previous sections. There are 90, 101, and 152 parameters in Kissat 3.0.0, 3.1.1, and 4.0.1, respectively, and all of them are integer and finite. If a solver is run with the option `--range`, it prints the left bound l, the default value d, and the right bound r of all parameters. It is clear that a domain's size is $n = r - l + 1$. In Kissat's configuration space, n varies from 2 to 2^{31}.

We decided to keep a configuration space as small as possible since the objective function is costly (i.e., much computational resources are needed to calculate it). First, we took 88 parameters which present in all three Kissat's versions since a parameter that remains in different versions of a solver for years can be considered important. Second, we excluded 49 of them from the consideration since in preliminary experiments they did not affect Kissat's performance. Some of them alter the output statistics, while other are redundant. Third, we additionally reduced parameters' domains manually to have $n \leq 10$ in each domain. Note that all left bounds, right bounds, and default values were not changed. The chosen 39 parameters and their domains are presented in Appendix B. The corresponding configuration space consists of 3.45e+28 parameter configurations, so it is still not feasible to try them all.

7 Preimage attacks by parameterized Kissat

In this section, the approach proposed in Section 6 is applied to parameterize Kissat on intermediate inverse problems to invert 24-round SHA-1, 29-step MD5, and 19-round SHA-256.

7.1 Experimental setup

The algorithm configuration problem was considered as follows:

1. ParamILS 2.3.8 as a configurator;

2. a set of SAT instances that encode intermediate inverse problems (specific for each compression function, see the following subsections) as a training set;
3. Kissat 4.0.1 as a target algorithm;
4. the Kissat's configuration space described in Subsection 6.3;
5. the default Kissat's configuration as a start configuration;
6. PAR1000 as an objective function;
7. a time limit for SAT instances (specific for each compression function);
8. a time limit of 72 hours for ParamILS.

A computer equipped with two 96-core CPUs AMD EPYC 9654 and 768 Gb RAM was used in these experiments. ParamILS 2.3.8 is a single-threaded application. Besides the aforementioned inputs, it takes a random seed to initialize the randomization. On the first training set, ParamILS was run 192 times with the random seed varied from 0 to 191 to employ all CPU cores. In case of the second training set, random seed was varied from 192 to 383, then from 384 to 575 for the third set.

In preliminary experiments, it turned out that a training set must contain at least 10 instances to find a new configuration that works better on harder intermediate inverse problems. In the following subsections, each training set contains 16 instances.

According to the predictions from Section 5, three main considered problems in ascending order of hardness are: inverting 24-round SHA-1; inverting 29-round MD5; inverting 19-round SHA-256. That is why below they are solved in this order.

7.2 Inverting 24-step SHA-1

For SHA-1, the first training set consisted of 16 CNFs: the last 15 intermediate inverse problems between rounds 21-22 and the one for 22 rounds, all for 1-hash. These CNFs were chosen because the objective function value on this set is reasonable: it is 43 minutes on 1 CPU core with a time limit of 1 day when the default configuration is applied. For comparison, on the first 16 intermediate inverse problems between rounds 22-23 the objective function value is 1 day 19 hours.

As mentioned above, ParamILS was run 192 times using different random seeds. A time limit of 33 minutes for Kissat was used since this was sufficient to solve any instance from the set using the default configuration. Within ParamILS's time limit of 3 days, the best result was found on seed 69. In total, 3,085 configurations were tried on this seed, the best configuration was updated 5 times, and on the final one the objective function value was 10 minutes, so a 4X speed-up was achieved.

We decided to parameterize Kissat one more time on the second training set consisted of the first 16 intermediate inverse problems between rounds 22-23. The best configuration found in the previous experiment was used as a start configuration in this experiment. The objective function value on the start configuration was 11 hours 17 minutes (compared to 1 day 19 hours on the default configuration). A time limit for Kissat was set to 2 hours 30 minutes. ParamILS was run 192 times as earlier. This time seed 279 gave the best result: 321 configurations were tried, the best configuration was updated once, and the corresponding objective function value was 4 hours 30 minutes. The speed-up was about 2X/10X compared to the start/default

configuration. Values of 28 parameters were changed compared to the default one. The best found configuration is presented in Appendix C.

The parameterized Kissat was run with a time limit of 1 day on 97 CNFs between rounds 21-24 as it was done for the default Kissat in Subsection 5.2. The results are presented in Figure 4. The default version solved 69 problems, while the parameterized one solved 74 problems and on most of them it was faster. More importantly, the parameterized Kissat solved 9 intermediate inverse problems between rounds 23-24. In particular, it inverted (23 $\frac{26}{32}$)-round SHA-1. In contrast, the default version solved only 4 intermediate inverse problems, yet at most (23 $\frac{22}{32}$)-round SHA-1 was inverted. Therefore, it is clear that the parameterized version outperforms the default one. Similarly to Subsection 5.2, we used regression analysis to predict a runtime needed to invert 24 rounds, see Figure 4. Recall that despite the exponential behavior of the runtime, the regression is shown as a line since y-axis is in the logarithmic scale. According to the prediction, it would take the parameterized Kissat 6 days to invert 24-round SHA-1, yet the default version would solve it in 9 days.

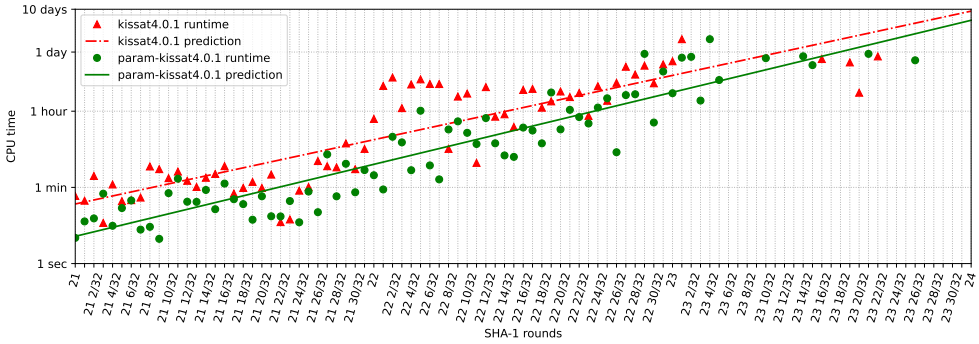


Fig. 4: Comparison of the default Kissat with its parameterized version on inverse problems for rounds 21-24 of SHA-1, 1-hash. The y-axis is in the logarithmic scale.

To invert 24-round SHA-1, we chose the Cube-and-Conquer approach — a given problem is split via a lookahead solver into subproblems, which are solved by a CDCL solver [41]. A proper splitting is crucial for Cube-and-Conquer. Usually this is managed by choosing a value of a cutoff threshold that regulates switching from splitting to forming a subproblem for a CDCL solver.

In particular, we applied a Cube-and-Conquer-based algorithm EnCnC [20]. It was EnCnC that inverted 1-hash produced by the 43-step MD4 and 28-step MD5 compression functions for the first time [20]. Briefly, the algorithm generates a set of promising cutoff thresholds and then for each threshold a random sample of subproblems is formed and all of them are solved by a CDCL solver with a time limit. In the present paper we decided to employ EnCnC in an incomplete SAT solving mode aimed at SAT instances with a huge number of satisfying assignments. The following inputs were given to EnCnC:

- lookahead solver March_cu [42];

- the parameterized Kissat;
- `nstep` = 1;
- `minc` = 5000;
- `maxc` = 1000000;
- `minr` = 10;
- N = 1000;
- `maxst` = 5000 seconds;
- `cores` = 192;
- `mode` = `incomplete-solving`.

The time limit 5,000 seconds (1 hour 23 minutes) for instances from samples is standard for SAT Competitions [43], so a state-of-the-art CDCL solver is designed to do its best within it. The remaining values of EnCnC's inputs were chosen to form at least 23 cutoff thresholds. The reason is that at most 23,224 instances can be solved with a time limit 5,000 seconds within a week on 192 CPU cores, yet we could not afford spending more resources on a single computational task.

A preimage for 24-round SHA-1 was found by EnCnC in 3 hours 53 minutes. The parameterized Kissat was interrupted on 384 instances because of the time limit and solved one instance in 59 minutes. Table 2 presents the found preimage, where the first row contains values of $M[0], M[1], M[2], M[3]$, the second row contains that of $M[4], M[5], M[6], M[7]$ and so on.

Table 2: A preimage of 160 one-bits produced by 24-round SHA-1 compression function.

0xd90b5a14	0x9bbec456	0x72f7a3db	0xddf48dd0
0xf869acc6	0x77dc416c	0x3bf8499c	0x5ecfd3a6
0x740f6a1f	0xa1d597b	0x9f794868	0x14c40240
0xa1ec0639	0x5ea901b7	0xc6e4df8c	0xe8a841de

EnCnC was also run with the default Kissat as a CDCL solver and as a result a preimage was found in 8 hours 30 minutes. It means that the parameterized Kissat outperforms the default one on SHA-1-based subproblems generated by Cube-and-Conquer.

Since 24-round SHA-1 was inverted in a reasonable time, we also attempted solving three intermediate inverse problems for (24 $\frac{2}{32}$)-round, (24 $\frac{4}{32}$)-round, and (24 $\frac{6}{32}$)-round SHA-1. For convenience, they are further called 24.0625-round, 24.125-round, and 24.1875-round SHA-1, respectively. EnCnC with the same parameterized Kissat was used for this purpose. To have at least 23 thresholds, `maxc` was increased to 5,000,000 for the first two problems, while for the third problem `minc`=20,000 and `maxc`=20,000,000 were used. As a result, preimages for 24.0625-round and 24.125-round SHA-1 were found in 1 day 20 hours and 1 day 4 hours. At the same time, no preimage for 24.1875-round SHA-1 was found within 7 days. Table 3 presents the preimage found for 24.125-round SHA-1.

Table 3: A preimage of 160 one-bits produced by 24.125-round SHA-1 compression function.

0xbc8bc542	0x6dc777da	0x5117e5b	0x90da0f3
0x1e4a35ca	0x2b5b1a80	0x6ad1d5d3	0x6cccfab
0x1cf68d15	0x2f409cf	0xff53d0ba	0x3bebeb5e
0xb1785cdd	0x30b21e0e	0x53425e6c	0x128b1a94

7.3 Inverting 29-step MD5

We applied ParamILS to MD5 two times. The first training set consisted of the last 15 intermediate inverse problems between steps 27-28 and the one for 28 steps. On the default configuration, the objective function value was 6 hours 11 minutes. In a ParamILS scenario, a time limit of 1 hour 23 minutes was set for Kissat. Within 3 days on 192 CPU cores, the best result was found on seed 63. In total, 786 configurations were tried on this seed, the best configuration was updated 3 times, and on the final one the objective function value was 1 hour 24 minutes, so a 4X speed-up was achieved.

The second training set consisted of the last ten intermediate inverse problems between steps 27-28, 28 steps, and the first five problems between steps 28-29. The best configuration found in the previous experiment was used as a start configuration in this experiment. The objective function value on the start configuration was 11 hours 57 minutes (compared to 37 hours 16 minutes on the default configuration). A time limit for Kissat was set to 2 hours 47 minutes. ParamILS was run 192 times as earlier. This time seed 347 gave the best result: 230 configurations were tried on the seed, the best configuration was updated two times, and the final best objective function value was 3 hours 44 minutes. The speed-up was about 3X/10X compared to the start/default configuration. Values of 28 parameters were changed compared to the default one. The best found configuration is presented in Appendix C.

The parameterized Kissat was run with the time limit of 1 day on 67 instances between steps 27-29 of MD5 as it was done for the default Kissat in Subsection 5.2. The results are presented in Figure 5. The default version solved 43 problems, while the parameterized one solved 46 problems. Note that the parameterized Kissat solved 13 intermediate inverse problems between steps 28-29 while the default version solved only 10 of them. Similarly to Subsection 5.2, we used regression analysis to predict a runtime needed to invert 29 steps. According to the prediction, it would take the parameterized Kissat 18 days, yet the default version would solve it in 94 days.

EnCnC was run on 29-step MD5 with the following inputs: `minr`=1,000; `minc`=10,000; `maxc`=500,000; Kissat, parameterized on MD5. The remaining inputs are the same as were presented in the previous subsection. As a result, a preimage was found in 4 days 17 hours. This preimage is presented in Table 4. EnCnC was also run using the default Kissat, yet no preimage was found within 7 days.

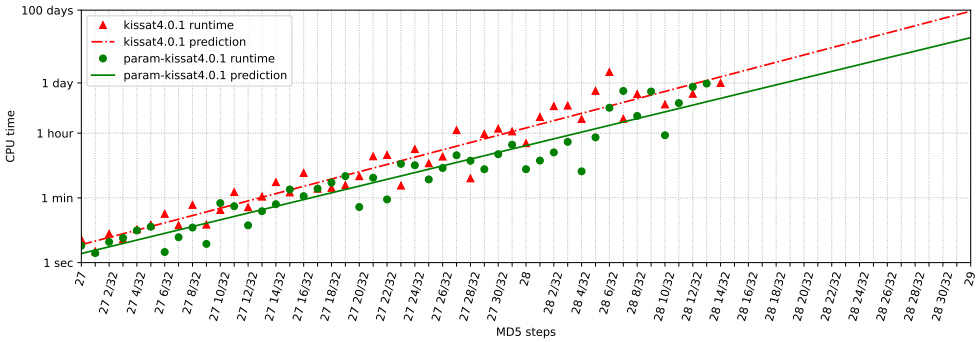


Fig. 5: Comparison of the default KISSAT with its parameterized version on intermediate inverse problems for steps 27-29 of MD5, 1-hash. The y-axis is in the logarithmic scale.

Table 4: A preimage of 128 one-bits produced by 29-step MD5 compression function.

0xca1ed26c	0x53005d49	0xeca03c53	0xa3d7ba2e
0xe6068fea	0x60367546	0xdda0dc31	0x2f8a23ec
0x9201cdb9	0x6e805f20	0xde276453	0xd40c08e8
0x4691e663	0x63f8a4cc	0x8f2807fb	0xfa40d5d

7.4 Inverting 19-round SHA-256

According to the prediction from Subsection 5.2, inverting 19-round SHA-256 is the hardest considered computational problem, so three PARAMILS-related experiments were conducted on 192 CPU cores. The results are presented in Table 5. In the first column, numbers of intermediate inverse problems between rounds 18-19 are specified, all for 1-hash.

Table 5: PARAMILS results obtained on intermediate inverse problems between rounds 18-19 of SHA-256.

Training set	KISSAT time limit	Best seed value / configurations	Best function value
Problems 1-16	5 h 19 min	118 / 3,967	2 min 12 sec
Problems 3-18	1 h 27 min	229 / 4,267	6 min
Problems 4-19	3 h 3 min	499 / 3,970	37 min

The first configuration from the table provides a 32X speed-up on the first training set when compared to the default configuration. As for the other two configurations, a speed-up is hard to calculate directly since on the default configuration intermediate inverse problems with $j = 17, 19$ were not solved within the time limits. Consider only problems which were solved on the default configuration. Then a 13X speed-up

on the second training set was achieved since on the default/second configuration the objective function value was 1 h 10 min / 6 minutes 29 seconds. As for the third configuration from the table, the speed-up was about 3X since the objective function value on the default/third configuration was 1 h 10 min / 25 minutes. The third configuration is presented in Appendix C. Compared to the default configuration, values of 31 parameters out of 39 were changed.

The parameterized KISSAT was run with a time limit of 1 day on 33 problems between rounds 18-19. The results are presented in Figure 6. The default version solved 17 problems, while the parameterized one solved 22 of them. We used regression analysis to predict a runtime needed to invert 19 rounds. According to the prediction, it would take the parameterized KISSAT 19 days, yet the default version would solve it in 189 days.

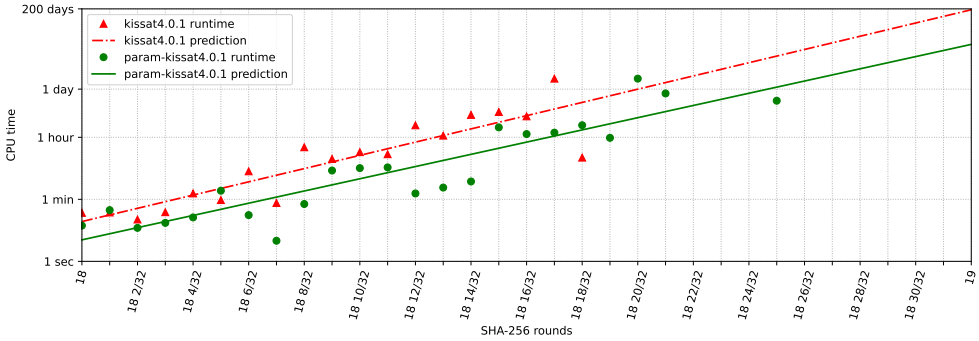


Fig. 6: Comparison of the default KISSAT with its parameterized version on intermediate inverse problems for rounds 18-19 of SHA-256, 1-hash. The y-axis is in the logarithmic scale.

ENCNC was run on 19-round SHA-256 with the following inputs: `minr`=100; `minc`=10,000; `maxc`=1,000,000; the parameterized KISSAT (using the third configuration). No preimage was found within 7 days on 192 CPU cores. Then we considered three intermediate inverse problems for $j = 20, 24, 28$ that were solved by neither the default nor the parameterized KISSAT. ENCNC's inputs for $j = 20, 24$ were `minr`=100, `minc`=10,000, `maxc`=100,000, while for $j = 28$ `minc` and `maxc` were increased to 50,000 and 500,000, respectively. The results are presented in Table 6. ENCNC with the default KISSAT could not solve the problem for $j = 28$, while ENCNC with the parameterized KISSAT solved all three problems. It is clear that the parameterized version significantly outperformed the default one on these problems.

Then ENCNC with the parameterized KISSAT was run on $j = 29$ with the same inputs as for $j = 28$. This problem was successfully solved as well (see the runtime in the table). It turned out, that the found preimage of $(18\ \frac{29}{32})$-round SHA-256 is also a preimage of 19-round SHA-256. Subsection 3.1 mentioned that there is a nonzero probability that this could happen. In this case the probability was $\frac{1}{8}$. Note that in case

of $j = 20, 24, 28$ the probability was $\frac{1}{4096}$, $\frac{1}{256}$, and $\frac{1}{16}$, and no preimage for 19-round SHA-256 was found.

Table 7 presents the found preimage of 19-round SHA-256. EnCnC with the default Kissat was also run on $j = 29$, but no preimage was found within 7 days.

Table 6: EnCnC runtimes on SHA-256 intermediate inverse problems between rounds 18 and 19. The value "-" means that nothing was found within 7 days.

Conquer solver	$j = 20$	$j = 24$	$j = 28$	$j = 29$	$j = 32$
Default Kissat	1 h 38 min	61 h 45 min	67 h 32 min	-	-
Parameterized Kissat	8 min	4 h 16 min	7 h 13 min	18 h 33 min	-

Table 7: A preimage of 256 one-bits produced by 19-round SHA-256 compression function.

`0x61ef77ad`	`0xf404a654`	`0x32332d66`	`0x979944c3`
`0xb0a84a7c`	`0x9ed99956`	`0x76979691`	`0x9d4f3e47`
`0x325cd2e1`	`0x81739d58`	`0xb3c87ef0`	`0x2f6f2587`
`0x6a7be4d4`	`0xb4dd5faf`	`0xc5c05d37`	`0xa00c6b8f`

We also tried to invert 29-step MD5, 24-round SHA-1, and 19-round SHA-256 on the same computer by off-the-shelf parallel SAT solvers: seven solvers from the SAT Competition 2024 and ALIAS solver [44]. The last one was chosen because it has shown promising results on cryptographic problems. None of three mentioned inverse problems was solved by these solvers.

8 Discussion

According to the results, the hypothesis proposed in Section 6 is experimentally confirmed, and it gives the fourth take away: *parameterizing a state-of-the-art CDCL solver's parameters on simple intermediate inverse problems allows faster solving of hard intermediate inverse problems.*

Three found configurations for MD5, SHA-1, and SHA-256 differ a lot, but the following values exist in all of them: `stable`=2, `tier1`=5. Note that the default Kissat's values are different.

According to the results, it is realistic to invert 25-round SHA-1 compression function by allocating more computational resources for both parameterizing and solving. On the other hand, it is likely that additional effective algorithmic techniques are required to invert 30-step MD5 and 20-round SHA-256 compression functions.

8.1 Real runtime vs predicted runtime

Since preimages of 24-round SHA-1, 29-step MD5, and 19-round SHA-256 compression functions were found, the corresponding predicted runtimes can be validated. Recall

that predicted runtimes for the default Kissat were calculated in Subsection 5.2 (see Figure 3), while that of the parameterized Kissat was done in Section 7 (see Figures 4, 5, 6).

As predicted, the default Kissat would find a preimage for 24-round SHA-1 in 9 days on 1 CPU core. Then in Subsection 7.2 EnCnC based on the default Kissat did it in 8 hours 30 minutes on 192 CPU cores. If 1 CPU core had been used, then EnCnC would have done it in 68 days. It means that the real runtime turned out to be about 6.8X higher that the predicted runtime. Details on the remaining cases are presented in Table 8.

Table 8: Comparison of predicted runtimes with real runtimes.

Problem	Kissat	Predicted, 1 core	Real, 192 cores	Real, 1 core	Ratio
24-round SHA-1	default	9 d	8 h 30 min	68 d	6.8X
	param.	6 d	3 h 53 min	31 d	5.2X
29-step MD5	param.	18 d	4 d 17 h	904 d	50.2X
19-round SHA-256	param.	19 d	18 h 33 min	148 d	7.8X

According to the table, the real runtimes in case of SHA-1 and SHA-256 are 5-8X higher that the predicted runtimes, while MD5's real time is 50X higher. On the one hand, the predictions are overly optimistic. On the other hand, even in the present form these predicted runtimes can be considered as lower bounds that can be useful to allocate required computational resources. Also, such predictions can help comparing encodings, solvers, and parameter configurations.

8.2 Second preimage attacks

The found preimages differ from those published in [9], so in the present paper for the first time second preimage attacks (see Subsection 2.2) on 29-step MD5 and 24-round SHA-1 compression functions are performed in addition to the corresponding preimage attacks.

8.3 Reproducibility

The presented preimages were hard to find, but their verification via direct computations takes a fraction of a second. The correctness was verified by modifying reference implementations from [16, 21, 24] written in C. First, padding (see Section 2), as well as all unnecessary operations (steps, rounds) in a compression function must be deleted. Note that the incrementation stage, which is performed after all remaining operations, is not deleted. Then a preimage should be given as an input of the modified compression function. As a result, 128 one-bits, 160 one-bits, and 256 one-bits are produced for MD5, SHA-1, and SHA-256, respectively.

8.4 Comparison with other methods

As mentioned in Section 2, theoretical preimage attacks on 64-round (full) MD5 cryptographic hash function, 56-round (reduced) SHA-1 cryptographic hash function, and 52-round (reduced) SHA-256 compression function exist [23, 25, 45]. It means that these papers proposed theoretical algorithms for preimage attacks with lower complexity than brute force. It seems that it is in principle possible to modify these algorithms to mount practical attacks on the reduced versions of compression functions, for which record results were obtained in the present paper. However, such modifications were not made in the course of the present research since they are highly nontrivial.

Consider computational resources required to mount preimage attacks in the previous sections.

Table 9: Computational resources required to mount preimage attacks in the present paper.

MD5		
27 steps 4 sec, 1 core	28 steps 32 min, 1 core	29 steps 4 d 17 h, 192 cores
SHA-1		
22 rounds 40 min, 1 core	23 rounds 14 h 39 min, 1 core	24 rounds 3 h 53 min, 192 cores
SHA-256		
17 rounds 1 sec, 1 core	18 rounds 25 sec, 1 core	19 rounds 18 h 33 min, 192 cores

As shown in the table, the computational costs increase sharply on the last considered operation (round or step). If the cost increases that sharply for all following operations, then the SAT approach will require more resources than brute force to mount preimage attacks on the aforementioned versions, for which theoretical attacks were held. It is another argument in favor of mounting practical attacks based on those theoretical algorithms.

9 Related work

SAT has been applied to cryptographic hash functions of the MD family as follows. In [46], practical collision attacks on MD4 and MD5 were performed. In [1], 39-step MD4 was inverted, while for steps 40-43 it was done in [47]. 28-step MD5 was inverted in [2, 20].

SAT has been also applied to the SHA family. 22-round SHA-1 was inverted in [3, 6], then for the 23-round version it was done in [2, 27]. In [7], a weakened 24-round SHA-1 was inverted, such that the number of known hash bits was reduced from 160 to 128. In [6], full SHA-1 was inverted, but most message bits were assigned randomly, yet in [8] a similar result was achieved for SHA-256. In [4], 16-round SHA-256 and 2-round SHA-3 were inverted. A semi-free-start collision for 38-round SHA-256 was

found in [48]. The first collision for full SHA-1 was found in [22] and partially it was done by a SAT solver.

The following hard mathematical problems have been solved via Cube-and-Conquer in the past decade: the Boolean Pythagorean Triples problem [49], the Schur number five problem [50], Lam's problem [51], Keller's Conjecture [52]. Cube-and-Conquer is also applicable to physics: the lower bound for the Minimum Kochen–Specker Problem was improved in [53].

In the present paper, a costly black-box objective function is minimized to solve hard cryptanalysis problems by a parallel SAT solver. Similar objective functions were minimized in [54, 55] with the same goal.

10 Conclusion

This paper proposed a new type of intermediate inverse problems between any two consecutive operations i and $i+1$ of a cryptographic hash function from a wide class. First, these problems are useful to make some progress if i operations can be inverted in a reasonable time while the inversion of $i+1$ steps is infeasible. Second, simple intermediate problems can be used to parameterize a CDCL solver in a way it can solve previously unattainable problems. Third, if some intermediate inverse problems are solved between operations i and $i+1$, a runtime prediction for inverting $i+1$ steps can be calculated. Fourth, in some cases it is not even needed to invert $i+1$ operations directly since a solution of an intermediate inverse problem between operations i and $i+1$ can be simultaneously a preimage of $i+1$ operations. The main result of the paper is inverting 29-step MD5, 24-round (and 24.125-round) SHA-1, and 19-round SHA-256 compression functions for the first time, thus making a clear progress in this area. In the future we are going to apply the proposed approach to analyze other cryptographic hash functions.

Acknowledgements. The author thanks anonymous reviewers for valuable and thorough comments. The author also thanks Stepan Kochemazov for fruitful discussions. The author is grateful to the Irkutsk Supercomputer Center of SB RAS for providing computational resources.

Author contribution. The author confirms the sole responsibility for the conception of the study, presented results and manuscript preparation.

Funding. This work was funded by the Ministry of Science and Higher Education of the Russian Federation, project No. 121041300065-9.

Data availability. All CNFs used in our experiments are available online at https://github.com/olegzaikin/sha1-sat.

Code availability. Programs for generating CNFs are available online at https://github.com/olegzaikin/sha1-sat and https://github.com/olegzaikin/SAT-encoding. Scripts for producing configuration spaces for Kissat are available online at https://github.com/olegzaikin/paramsat.

Materials availability. Not applicable.

Declarations

Conflict of interest. The author has no competing interests or conflicts to declare.

Ethics approval. Not applicable.

Consent to participate. Not applicable.

Consent for publication. Not applicable.

A Random outputs

Table 10 contains 8 random sequences which were used as outputs to generate SAT instances.

Table 10: Eight random 256-bit sequences in the hexadecimal form.

0xdf929bb8b4136b686067d625801408a392e0d3f2a7b88ecacf4380666a43bcc7
0xa4ea4f9eb6409eaa89c0e5f8e5c059c5e1df8669b90acd63b186649dd2c40617
0x1f8a2388ee6ddc4d9b5656b27b870e3f0ef7b471464f751f92117bec28aa7c49
0xfc9de05627af825351b8c821a78eae2da94099b7730405396afbadcb7291fe15
0x983efa3a4c5517f55edb1856cc2517d09548fc84b2bf3512f31a4c0b193a2102
0xe536d6eac5d9d6cbb744e403db6fe028778172ad5f2a76501b8739b2f3bfb533
0xdd3a5442090eabc9152dbabad73c456caaa604f3f83498e4e16dd1bc84ee4b54
0x485b34e25b358b8bbd8c459f9091052d80fe02a7cbeff29f98430bd0d4a2397e

B Varied Kissat parameters and their Domains

Table 11 contains 39 chosen Kissat 4.0.1 parameters, their domains, and default values. Domains of all parameters but `backbone`, `shrink`, `stable`, and `target` were constructed manually.

C Found configurations

This section contains the final best configurations found for SHA-1, MD5, and SHA-256. The configurations are presented by listing values of all 39 parameters specified in Table 11.

SHA-1: 1, 2147483647, 2147483647, 10000000, 2147483647, 1, 25, 25, 0, 16, 10000000, 2000, 1, 10000, 100000, 10, 1000, 100000000, 8192, 1073741824, 1000, 10, 3, 2, 1, 25, 1000, 2147483647, 2147483647, 1000, 100000, 32768, 3, 2147483647, 2, 5, 3, 10, 0.

MD5: 1, 1, 1000, 10000000, 2147483647, 5, 25, 50, 1024, 0, 100000, 2147483647, 16, 1000, 100000, 10, 1000, 10, 1024, 1024, 1, 25, 3, 2, 100, 50, 1000, 1000, 2147483647, 100000, 0, 10, 3, 2147483647, 1, 5, 3, 6, 50.

SHA-256: 2, 100000, 10000000, 10, 100, 10, 1, 2, 2147483647, 1024, 10000000, 1048576, 2, 10000, 1000000, 0, 10000, 100000000, 128, 1024, 1, 5, 0, 2, 1, 2, 1, 2147483647, 0, 0, 2147483647, 10000000, 10000000, 10000000, 2, 5, 1000, 3, 0.

Table 11: Chosen parameters, domains, and default values.

Parameter	Domain	Default value
backbone	0, 1, 2	1
backbonerounds	1, 100, 100000, 10000000, 2147483647	100
bumpreasonslimit	1, 10, 1000, 100000, 10000000, 2147483647	10
bumpreasonsrate	1, 10, 1000, 100000, 10000000, 2147483647	10
chronolevels	0, 100, 100000, 10000000, 2147483647	100
compactlim	0, 1, 2, 5, 10, 25, 50, 100	10
decay	1, 2, 5, 10, 25, 50, 100, 200	50
definitioncores	1, 2, 5, 10, 25, 50, 100	2
definitionticks	0, 1024, 1000000, 2147483647	1000000
eliminatebound	0, 2, 16, 128, 1024, 8192	16
eliminateclslim	1, 100, 100000, 10000000, 2147483647	100
eliminateocclim	0, 2000, 1048576, 2147483647	2000
eliminaterounds	1, 2, 16, 128, 1024, 10000	2
emafast	10, 33, 100, 1000, 10000, 100000, 1000000	33
emaslow	100, 1000, 10000, 100000, 1000000	100000
mineffort	0, 10, 1000, 100000, 10000000, 2147483647	10
minimizedepth	1, 10, 100, 1000, 10000, 100000, 1000000	1000
modeinit	10, 1000, 100000, 10000000, 100000000	1000
reluctantint	2, 16, 128, 1024, 8192, 32768	1024
reluctantlim	0, 1024, 1048576, 1073741824	1048576
restartint	1, 10, 100, 1000, 10000	1
restartmargin	0, 1, 2, 5, 10, 25	10
shrink	0, 1, 2, 3	3
stable	0, 1, 2	1
substituteeffort	1, 10, 100, 1000	10
substituterounds	1, 2, 5, 10, 25, 50, 100	2
subsumeclslim	1, 1000, 1048576, 2147483647	1000
subsumeocclim	0, 1000, 1048576, 2147483647	1000
sweepclauses	0, 1024, 1048576, 2147483647	1024
sweepdepth	0, 2, 1000, 100000, 10000000, 2147483647	2
sweepfliprounds	0, 1, 1000, 100000, 10000000, 2147483647	1
sweepmaxclauses	2, 10, 1000, 32768, 10000000, 2147483647	32768
sweepmaxdepth	1, 3, 1000, 100000, 10000000, 2147483647	3
sweepvars	0, 256, 100000, 10000000, 2147483647	256
target	0, 1, 2	1
tier1	1, 2, 5, 10, 25, 50, 100	2
tier2	1, 3, 6, 10, 25, 50, 100, 200, 500, 1000	6
vivifytier1	0, 1, 3, 6, 10, 25, 50, 100	3
vivifytier2	0, 1, 3, 6, 10, 25, 50, 100	3

References

[1] De, D., Kumarasubramanian, A., Venkatesan, R.: Inversion attacks on secure hash functions using SAT solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4501, pp. 377–382. Springer, Cham (2007). https://doi.org/10.1007/978-3-540-72788-0_36

[2] Legendre, F., Dequen, G., Krajecki, M.: Encoding hash functions as a SAT problem. In: IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012, pp. 916–921. IEEE Computer Society, USA (2012). https://doi.org/10.1109/ICTAI.2012.128

[3] Srebrny, M., Srebrny, M., Stepien, L.: SAT as a programming environment for linear algebra and cryptanalysis. In: International Symposium on Artificial Intelligence and Mathematics, ISAIM 2008, Fort Lauderdale, Florida, USA, January 2-4, 2008 (2008)

[4] Homsirikamol, E., Morawiecki, P., Rogawski, M., Srebrny, M.: Security margin evaluation of SHA-3 contest finalists through SAT-based attacks. In: Cortesi, A., Chaki, N., Saeed, K., Wierzchon, S.T. (eds.) Computer Information Systems and Industrial Management - 11th IFIP TC 8 International Conference, CISIM 2012, Venice, Italy, September 26-28, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7564, pp. 56–67. Springer, Cham (2012). https://doi.org/10.1007/978-3-642-33260-9_4

[5] Marques-Silva, J., Sakallah, K.: GRASP: A search algorithm for propositional Satisfiability. IEEE Trans. Computers **48**(5), 506–521 (1999) https://doi.org/10.1109/12.769433

[6] Nossum, V.: SAT-based preimage attacks on SHA-1. Master's thesis, University of Oslo, Department of Informatics (2012)

[7] Bellini, E., Piccoli, A.D., Makarim, R.H., Polese, S., Riva, L., Visconti, A.: New records of pre-image search of reduced SHA-1 using SAT solvers. In: Giri, D., Choo, K.R., Ponnusamy, S., Meng, W., Akleylek, S., Maity, S.P. (eds.) Proceedings of the Seventh International Conference on Mathematics and Computing - ICMC 2021, Shibpur, India. Advances in Intelligent Systems and Computing, vol. 1412, pp. 141–151. Springer, Cham (2021). https://doi.org/10.1007/978-981-16-6890-6_11

[8] Choo, D., Soos, M., Chai, K.M.A., Meel, K.S.: Bosphorus: Bridging ANF and CNF solvers. In: Teich, J., Fummi, F. (eds.) Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019, pp. 468–473. IEEE, USA (2019). https://doi.org/10.23919/DATE.2019.8715061

[9] Zaikin, O.: Inverting step-reduced SHA-1 and MD5 by parameterized SAT solvers. In: Shaw, P. (ed.) 30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2-6, 2024, Girona, Spain. LIPIcs, vol. 307, pp. 31–13119. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2024). https://doi.org/10.4230/LIPICS.CP.2024.31

[10] Biere, A., Heule, M., Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability - Second Edition. Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press, Amsterdam (2021). https://doi.org/10.3233/FAIA336

[11] Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA, pp. 151–158 (1971). https://doi.org/10.1145/800157.805047

[12] Massacci, F., Marraro, L.: Logical cryptanalysis as a SAT problem. J. Autom. Reasoning **24**(1/2), 165–203 (2000) https://doi.org/10.1023/A:1006326723002

[13] Menezes, A., Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, USA (1996). https://doi.org/10.1201/9781439821916

[14] Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO'89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer, Cham (1989). https://doi.org/10.1007/0-387-34805-0_21

[15] Damgård, I.: A design principle for hash functions. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO'89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. Lecture Notes in Computer Science, vol. 435, pp. 416–427. Springer, Cham (1989). https://doi.org/10.1007/0-387-34805-0_39

[16] Rivest, R.L.: The MD5 message-digest algorithm. RFC **1321**, 1–21 (1992) https://doi.org/10.17487/RFC1321

[17] Rivest, R.L.: The MD4 message digest algorithm. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. Lecture Notes in Computer Science, vol. 537, pp. 303–311. Springer, Cham (1990). https://doi.org/10.1007/3-540-38424-3_22

[18] Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494, pp. 19–35. Springer, Cham (2005). https://doi.org/10.1007/11426639_2

[19] Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 134–152. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_8

[20] Zaikin, O.: Inverting cryptographic hash functions via Cube-and-Conquer. J. Artif. Intell. Res. **81**, 359–399 (2024) https://doi.org/10.1613/JAIR.1.15244

[21] Eastlake, D.E., Jones, P.E.: US secure hash algorithm 1 (SHA1). RFC **3174**, 1–22 (2001) https://doi.org/10.17487/RFC3174

[22] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 570–596. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_19

[23] Espitau, T., Fouque, P., Karpman, P.: Higher-order differential meet-in-the-middle preimage attacks on SHA-1 and BLAKE. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 683–701. Springer, Cham (2015). https://doi.org/10.1007/978-3-662-47989-6_33

[24] U.S. Department of Commerce, National Institute of Standards and Technology. Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-3 (2008). http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[25] Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. In: Canteaut, A. (ed.) Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. Lecture Notes in Computer Science, vol. 7549, pp. 244–263. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_15

[26] Morawiecki, P., Srebrny, M.: A SAT-based preimage analysis of reduced Keccak hash functions. Inf. Process. Lett. **113**(10-11), 392–397 (2013) https://doi.org/10.1016/J.IPL.2013.03.004

[27] Nejati, S., Liang, J.H., Gebotys, C.H., Czarnecki, K., Ganesh, V.: Adaptive restart and CEGAR-based solver for inverting cryptographic hash functions. In: Paskevich, A., Wies, T. (eds.) Verified Software. Theories, Tools, and Experiments - 9th International Conference, VSTTE 2017, Heidelberg, Germany, July 22-23, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10712, pp. 120–131. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72308-2_8

[28] Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni-Vincentelli, A.L.: Logic Minimization Algorithms for VLSI Synthesis. The Kluwer International Series in Engineering and Computer Science, vol. 2. Springer, New York (1984). https://doi.org/10.1007/978-1-4613-2821-6

[29] Nejati, S., Ganesh, V.: CDCL(Crypto) SAT solvers for cryptanalysis. In: Pakfetrat, T., Jourdan, G., Kontogiannis, K., Enenkel, R.F. (eds.) Proceedings of the

29th Annual International Conference on Computer Science and Software Engineering, CASCON 2019, Markham, Ontario, Canada, November 4-6, 2019, pp. 311–316 (2019). https://doi.org/10.5555/3370272.3370307

[30] Otpuschennikov, I., Semenov, A., Gribanova, I., Zaikin, O., Kochemazov, S.: Encoding cryptographic functions to SAT using TRANSALG system. In: Kaminka, G.A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum, V., Dignum, F., Harmelen, F. (eds.) ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016). Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1594–1595. IOS Press, Amsterdam (2016). https://doi.org/10.3233/978-1-61499-672-9-1594

[31] Semenov, A., Otpuschennikov, I., Gribanova, I., Zaikin, O., Kochemazov, S.: Translation of algorithmic descriptions of discrete functions to SAT with applications to cryptanalysis problems. Log. Methods Comput. Sci. **16**(1) (2020) https://doi.org/10.23638/LMCS-16(1:29)2020

[32] Biere, A., Faller, T., Fazekas, K., Fleury, M., Froleyks, N., Pollitt, F.: CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In: Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proceedings of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions. Department of Computer Science Report Series B, vol. B-2024-1, pp. 8–10 (2024)

[33] Ross, S.M.: Introduction to Probability and Statistics for Engineers and Scientists (2. Ed.). Academic Press, USA (2000)

[34] Hoos, H.H., Hutter, F., Leyton-Brown, K.: In: Biere, A., Heule, M., Maaren, H., Walsh, T. (eds.) Automated Configuration and Selection of SAT Solvers. Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 481–507. IOS Press, Amsterdam (2021). https://doi.org/10.3233/FAIA200995

[35] Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Paramils: An automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009) https://doi.org/10.1613/JAIR.2861

[36] Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers. Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer, Cham (2011). https://doi.org/10.1007/978-3-642-25566-3_40

[37] Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5732, pp. 142–157. Springer, Cham (2009). https://doi.

org/10.1007/978-3-642-04244-7_14

[38] Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H.H., Leyton-Brown, K.: The configurable SAT solver challenge (CSSC). Artif. Intell. **243**, 1–25 (2017) https://doi.org/10.1016/J.ARTINT.2016.09.006

[39] Ansótegui, C., Pon, J., Sellmann, M., Tierney, K.: Pydgga: Distributed GGA for automatic configuration. In: Li, C., Manyà, F. (eds.) Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12831, pp. 11–20. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80223-3_2

[40] Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., Hutter, F.: SMAC3: A versatile bayesian optimization package for hyperparameter optimization. J. Mach. Learn. Res. **23**, 54–1549 (2022)

[41] Heule, M., Kullmann, O., Wieringa, S., Biere, A.: Cube and Conquer: Guiding CDCL SAT solvers by lookaheads. In: Eder, K., Lourenço, J., Shehory, O. (eds.) Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC 2011, Haifa, Israel, December 6-8, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7261, pp. 50–65. Springer, Cham (2011). https://doi.org/10.1007/978-3-642-34188-5_8

[42] Heule, M., Maaren, H.: March_dl: Adding adaptive heuristics and a new branching strategy. J. Satisf. Boolean Model. Comput. **2**(1-4), 47–59 (2006) https://doi.org/10.3233/SAT190016

[43] Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M.: SAT competition 2020. Artif. Intell. **301**, 103572 (2021) https://doi.org/10.1016/J.ARTINT.2021.103572

[44] Kochemazov, S., Zaikin, O.: ALIAS: A modular tool for finding backdoors for SAT. In: Theory and Applications of Satisfiability Testing – SAT 2018: 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9–12, 2018, Proceedings, pp. 419–427. Springer, Berlin, Heidelberg (2018). https://doi.org/10.1007/978-3-319-94144-8_25

[45] Sasaki, Y., Komatsubara, W., Sakai, Y., Wang, L., Iwamoto, M., Sakiyama, K., Ohta, K.: Meet-in-the-middle preimage attacks revisited - new results on MD5 and HAVAL. In: Samarati, P. (ed.) SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013, pp. 111–122 (2013)

[46] Mironov, I., Zhang, L.: Applications of SAT solvers to cryptanalysis of hash functions. In: Biere, A., Gomes, C.P. (eds.) Theory and Applications of Satisfiability

Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4121, pp. 102–115. Springer, Cham (2006). https://doi.org/10.1007/11814948_13

[47] Zaikin, O.: Inverting 43-step MD4 via Cube-and-Conquer. In: Raedt, L.D. (ed.) Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022, pp. 1894–1900 (2022). https://doi.org/10.24963/IJCAI.2022/263

[48] Alamgir, N., Nejati, S., Bright, C.: SHA-256 collision attack with programmatic SAT. In: Brown, C.W., Kaufmann, D., Nalon, C., Steen, A., Suda, M. (eds.) Joint Proceedings of the 9th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 9th Satisfiability Checking and Symbolic Computation Workshop (SC-Square), 2024 Co-located with the 12th International Joint Conference on Automated Reasoning (IJCAR 2024), Nancy, France, July 2, 2024. CEUR Workshop Proceedings, vol. 3717, pp. 91–110 (2024)

[49] Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the Boolean Pythagorean triples problem via Cube-and-Conquer. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 228–245. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_15

[50] Heule, M.J.H.: Schur number five. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pp. 6598–6606 (2018). https://doi.org/10.1609/AAAI.V32I1.12209

[51] Bright, C., Cheung, K.K.H., Stevens, B., Kotsireas, I.S., Ganesh, V.: A SAT-based resolution of Lam's problem. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pp. 3669–3676 (2021). https://doi.org/10.1609/AAAI.V35I5.16483

[52] Brakensiek, J., Heule, M., Mackey, J., Narváez, D.E.: The resolution of Keller's conjecture. J. Autom. Reason. **66**(3), 277–300 (2022) https://doi.org/10.1007/S10817-022-09623-5

[53] Li, Z., Bright, C., Ganesh, V.: A SAT solver + computer algebra attack on the minimum Kochen-Specker problem. In: Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024, pp. 1898–1906 (2024)

[54] Semenov, A.A., Chivilikhin, D., Pavlenko, A., Otpuschennikov, I.V., Ulyantsev, V., Ignatiev, A.: Evaluating the hardness of SAT instances using evolutionary

optimization algorithms. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021. LIPIcs, vol. 210, pp. 47–14718. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Wadern (2021). https://doi.org/10.4230/LIPIcs.CP.2021.47

[55] Semenov, A., Zaikin, O., Kochemazov, S.: In: Pardalos, P.M., Rasskazova, V., Vrahatis, M.N. (eds.) Finding effective SAT partitionings via black-box optimization, pp. 319–355. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-66515-9_11